



Auteur : Raymond Moché

Mots-clefs : instructions conditionnelles, commandes « sqrt », « abs », « trier », « tirage_reel(p,a,b) », « plot(X) », « plot(X,Y,'r') », « plot(X,Y,'r+') », compteur, « %f », « %t », « reste », opérateurs logiques « & », « | » (« et » puis « ou »), test d'égalité numérique, « tic », « toc », instruction conditionnelle complexe « if (conditions) then (instructions) elseif (conditions) then (instructions) elseif (conditions) then (instructions) else (instructions) end ».

Référence :

- ✓ Livret de présentation de « scilab pour les lycées » (2010) et Mise à jour et compléments (mars 2011) :

<http://www.scilab.org/education/lycee/docs>

- ✓ Aide scilab 5.3.2

http://help.scilab.org/docs/5.3.2/fr_FR/index.html

Syntaxe de l'instruction conditionnelle « if-then-else »

- ✓ Instruction conditionnelle sans alternative

```
if conditions then
instructions;
end
```

- ✓ Instruction conditionnelle avec alternative simple

```
if conditions then
instructions;
else
instructions;
end
```

- ✓ Instruction conditionnelle avec alternative complexe

```
if conditions then
instructions;
elseif conditions then
instructions;
....
else
instructions;
end
```

Liste des exercices :

Énoncé n°1 : [*] Fonction valeur absolue.

Énoncé n°2 : [*] Ranger 2 nombres dans l'ordre croissant.

Énoncé n°3 : [*] Sauts de kangourous.

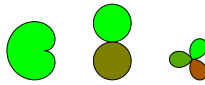
Énoncé n°4 : [*] Addition en base 2.

Énoncé n°5 : [**] L'année est-elle bissextile ?

Énoncé n°6 : [**] Calcul et représentation graphique de fréquences.

Énoncé n°7 : [**] Le point M est-il à l'intérieur du triangle ?

Énoncé n°8 : [**] Ce triangle est-il isocèle ?



Énoncé 1 [*] Fonction valeur absolue.

- 1 - Définir une fonction-scilab qui donne la valeur absolue de tout nombre réel
- à l'aide d'une instruction conditionnelle avec alternative,
- à l'aide d'une instruction conditionnelle sans alternative.
- 2 - En déduire la valeur absolue de -0.67 et de $\sqrt{3}$.

Mots-clefs : instruction conditionnelle, commandes « sqrt », « abs ».

Listing 1 – fonctions valeur absolue

```
// instruction conditionnelle sans alternative
clear
function y = valabs1(x)
  y=x;
  if x<0 then
    y=-x;
  end// pas d'alternative
endfunction
// instruction conditionnelle avec alternative
function y = valabs2(x)
  if x<0 then
    y=-x;
  else // alternative
    y=x;
  end
endfunction
```

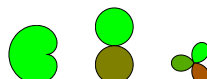
Ce qui donne

```
-->a1=valabs1(-0.87);a2=valabs2(-0.87);a=abs(-0.87);
-->b1=valabs1(sqrt(3));b2=valabs2(sqrt(3));b=abs(sqrt(3));
-->afficher(a1,a2,a,b1,b2,b)

1.7320508075689
1.7320508075689
1.7320508075689

0.87
0.87
0.87

-->
```



Énoncé 2 [*] Ranger 2 nombres dans l'ordre croissant.

Fonction-scilab : « ranger deux nombres réels donnés dans l'ordre croissant ».

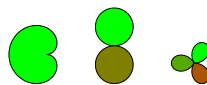
Mots-clefs : « instruction conditionnelle », commande « trier ».

Listing 2 – source scilab

```
// Fonction pour ranger x,y dans l'ordre croissant.
clear
function A=ranger(x,y);
    if x>y then// instruction conditionnelle sans alternative.
        z=x;
        x=y;
        y=z;
    end;
    A=[x,y];
endfunction

-->exec('/Users/raymondmoche/Magasin_scilab/304/304-fonction-ranger.sci', -1)
-->x=0.5;
-->y=-2;
-->Tableaurange=ranger(x,y);
-->afficher(Tableaurange);
- 2.    0.5
-->A=[0.5,-2];
-->B=trier(A);
-->afficher(B);
- 2.    0.5
-->
```

Évidemment, pour des choses aussi simples, il y a toujours une commande « scilab » qui convient : voir « trier » dans l'aide en ligne. Cette commande est très utilisée, par exemple en statistique, pour ordonner des échantillons.



Énoncé 3 [*] Sauts de kangourou.

Un kangourou fait habituellement des bonds de longueur aléatoire comprise entre 0 et 9 mètres.

1 - Combien de sauts devra-t-il faire pour parcourir 2000 mètres ?

2 - Fabriquer une fonction-scilab qui à toute distance d exprimée en mètres associe le nombre aléatoire N de sauts nécessaires au kangourou pour la parcourir.

3 - Calculer le nombre moyen de sauts effectués par le kangourou quand il parcourt T fois la distance d .

Mots-clefs : boucles « tant que » et « pour », affichage, « tic », « toc ».

Solution de la première question

Listing 3 – Nombre de sauts pour parcourir 2000 m.

```
D=0; // distance parcourue avant le premier saut.
N=0; // nombre de sauts effectués avant le premier saut.
while D<2000 then
    D=D+tirage_reel(1,0,9);
    N=N+1;
end
afficher('Le nombre de sauts nécessaires pour parcourir 2000 m
a été '+string(N)+' . La distance réelle parcourue a été '
+string(D)+' mètres.');
```

Exécution du script

Console Scilab

```
-->exec('/Users/raymondmoche/Magasin_scilab/304/304-Kangourou-2000m.sce', -1)

Le nombre de sauts nécessaires pour parcourir 2000 m a été 439. La dista
nce parcourue réellement a été 2003.2853616122 mètres.

-->
```

Solution des deuxième et troisième questions

Listing 4 – Nombre moyen de sauts.

```
d=input('La distance à parcourir est égale à ');
T=input('Le nombre de trajets est ');
tic();
function N=kangourou(d)
    D=0;
    N=0;
    while D<d then
        D=D+tirage_reel(1,0,9);
```

```

        N=N+1;
    end;
endfunction;
NSauts = [];
for j=1:T
    NSauts=[NSauts , kangourou(d)];
end;
Moyenne=sum(NSauts)/T;
afficher('Le nombre moyen de sauts quand le kangourou parcourt '
+string(T)+' fois '+string(d)+' mètres est '+string(Moyenne)+' ');
afficher('Durée du calcul : '+string(toc()));

```

Exécution du script (600 parcours de 2500 m)

Console Scilab

```

-->exec('/Users/raymondmoche/Magasin_scilab/304/304-MoySautsKang.sce', -1)
La distance à parcourir est égale à 2500
Le nombre de trajets est 600

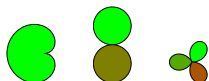
    Le nombre moyen de sauts quand le kangourou parcourt 600 fois 2500 mètres
        est 556.265.

    Durée du calcul : 16.059

-->

```

- ✓ On saisit les données puis on déclenche un chronomètre (commande « tic(); »).
- ✓ Définir une fonction-scilab n'est pas très utile ici.
- ✓ Une boucle « for » permet de faire le parcours T fois.
- ✓ On affiche le nombre moyen de sauts, on arrête le chronomètre puis on affiche le temps de calcul, qui est ici assez long.
- ✓ Il y a souvent plusieurs manières d'écrire un algorithme. Calculer les temps de calcul permet de progresser et aussi de se comparer aux informaticiens qui programment directement en C++, sauf erreur. Les comparaisons sont douloureuses.



Énoncé 4 ^[**] Addition en base 2

Soit deux entiers $n, m \geq 0$ notés en base 2 sous la forme $[a_0, \dots, a_{k-1}, a_k]$ et $[b_0, \dots, b_{k-1}, b_k]$. Cela signifie que ce sont deux suites de 0 et de 1 qui vérifient

$$n = a_0 \cdot 2^0 + a_1 \cdot 2^1 + \dots + a_k \cdot 2^k \text{ et } m = b_0 \cdot 2^0 + b_1 \cdot 2^1 + \dots + b_k \cdot 2^k$$

Il n'y a pas d'inconvénient à supposer que ces listes sont de même longueur, parce que l'on peut toujours ajouter des 0 à la liste la plus courte, si nécessaire. Par exemple, si on veut additionner 5 et 3 en base 2, on écrira 5 sous la forme $[1, 0, 1]$ et 3 sous la forme $[1, 1, 0]$ au lieu de $[1, 1]$. Le problème posé est de calculer leur somme en travaillant dans la base 2.

Mots-clefs : boucle « pour », instruction conditionnelle « if (conditions) then (instructions) elseif (conditions) then (instructions) elseif (conditions) then (instructions) else (instructions) end », opérateurs logiques « & », « | » (« et » puis « ou »).

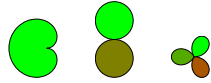
L'énoncé impose d'utiliser la table d'addition en base 2 :

+	0	1
0	0	1
1	1	10

On commence par additionner les coefficients de 2^0 . On conviendra qu'il y a une retenue, qui est 0. Pour les coefficients des puissances de 2 suivantes, s'il y en a, on ajoute les 2 coefficients et la retenue en se référant uniquement à la table d'addition (on ajoute les 2 premiers nombres puis le troisième au résultat). L'algorithme ci-dessous passe en revue tous les cas possibles. C'est donc élémentaire.

```
a=input('a=');
b=input('b=');// a et b sont deux listes de 0 et de 1 de même longueur.
function s=somme(a,b)
    r=0;// pas de retenue la première fois.
    s=[];// pour stocker les digits dyadiques de s.
    n=size(a,'c');
    for i=1:n
        if a(i)==0&b(i)==0&r==0 then
            s=[s,0];
            r=0;
        elseif (a(i)==1&b(i)==0&r==0)|(a(i)==0&b(i)==1&r==0)|(a(i)==0&b(i)==0&r==1) then
            s=[s,1];
            r=0;
        elseif (a(i)==1&b(i)==1&r==0)|(a(i)==1&b(i)==0&r==1)|(a(i)==0&b(i)==1&r==1) then
            s=[s,0];
            r=1;
        else
            s=[s,1];
            r=1;
        end
    end
    s=[s,r];
endfunction
s=somme(a,b);
```

```
afficher(s);
```



Énoncé 5 ^[**] L'année est-elle bissextile ?

- 1 - L'année n est-elle bissextile ?
- 2 - Soit n et m deux millésimes ($n \leq m$). Combien y a-t-il d'années bissextiles de l'année n à l'année m (les années n et m étant comprises) ?

Mots-clefs : fonction-scilab, instructions conditionnelles, compteur, « %f », « %t », « reste », opérateurs logiques.

Dans le script ci-dessous, on définit de deux manières différentes la fonction d'entrée n qui retourne « %t » ou « %f » suivant que l'année est bissextile ou non. Cela montre comment on utilise les opérateurs logiques (« | », « & », « == », etc).

Listing 5 – Années bissextiles

```
function [ truefalse]=estellebissextile(n)
  r=reste(n,4);
  s=reste(n,100);
  t=reste(n,400);
  truefalse=%f;
  if (((r==0)&(s>0))|(t==0)) then
    truefalse=%t;
  end;
endfunction;
function [ truefalse]=nestellepasbissextile(n)// fonction identique
à la précédente.
  r=reste(n,4);
  s=reste(n,100);
  t=reste(n,400);
  truefalse=%t;
  if ((r>0)|((s==0)&(t>0))) then
    truefalse=%f;
  end;
endfunction;
```

Le script téléchargeable « Bissextils.sce » contient le listing ci-dessus complété par des instructions d'affichage. Par exemple, 2012 est-elle bissextile ?

```
Console Scilab
-->exec('/Users/raymondmoche/Magasin_scilab/304/304_bissextils.sce', -1)
n=2012

  L'année est bissextile

  L'année est bissextile

-->
```

Pour traiter la seconde question, il suffit d'un compteur :

```

n=input('n=');
m=input('m=');// n<m.
C=0;// Initialisation du compteur
for i=n:m
    if estellebissextile(i)==%t then
        C=C+1;
    end
end
afficher('Le nombre d''années bissextilles recherch   est '+string(C));

```

Par exemple, trouver le nombre d'ann  es bissextilles de 2011    2407 donne, sur la console :

```

-->exec('/Users/raymondmoche/Magasin_scilab/304/304_bissextilles.sci', -1)

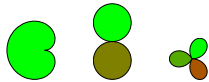
-->exec('/Users/raymondmoche/Magasin_scilab/304/304_Compteur-bissextilles.sce',
-1)
n=2011
m=2407

Le nombre d'ann  es bissextilles recherch   est 96

-->

```

(on charge dans « scilab » la fonction-scilab « estellebissextile », puis le script ci-dessus).



Énoncé 6 [**] Calcul et représentation graphique de fréquences.

On tire 10000 nombres au hasard dans l'intervalle $[0, 1]$. À chaque tirage, on se demande si l'événement

A : « le nombre tiré au hasard appartient à l'intervalle $[\frac{1}{7}, \frac{5}{6}]$ » est réalisé.

1 - Combien vaut la probabilité p de A ?

2 - Calculer la fréquence de réalisation de A après chaque tirage. Tracer le graphe des fréquences.

3 - Choisir la dernière fréquence calculée comme valeur approchée de p .

Mots-clefs : « instruction conditionnelle », boucle « pour », commandes « tirage_reel(p,a,b) », « plot(X) », « plot(X,Y,'r') », « plot(X,Y,'r+') », compteur.

Ce genre d'exercice se pose toutes les fois que l'on veut illustrer la convergence de la fréquence d'un événement vers sa probabilité, en référence à la loi des grands nombres, en Calcul des Probabilités. Le listing commenté « scilab » ci-dessous fournit une solution. Ce script est téléchargeable, sous le nom « Frequences.sce ». Il est suivi de commentaires concernant essentiellement les tracés.

Listing 7 – Calcul de fréquences et tracés

```
Tirages=tirage_reel(10000,0,1);// Tirages de 10000 nombres au hasard
// dans [0,1].
p=5/6-1/7;
afficher('La valeur exacte de p est '+string(p));
C=0;// Initialisation du compteur des réalisations de A.
Frequences=[];// Initialisation du vecteur-ligne des fréquences.
for i=1:10000
    if (1/7<=Tirages(1,i))&(Tirages(1,i)<=5/6) then
        C=C+1;// Si A se réalise, le compteur augmente de 1.
    end;
    Frequences=[Frequences,C/i];
end;
scf(0);// Ouverture d'une fenêtre graphique.
clf(0);// Nettoyage de cette fenêtre.
plot([0,10000],[p,p],'r');// Ligne horizontale d'ordonnée p, en rouge.
plot(Frequences);// Tracé de la ligne polygonale des fréquences.
// Il est sous-entendu (par défaut) que les abscisses varient
// de 1 à 10000 avec le pas 1.
afficher('La valeur approchée de p par les fréquences est '+
+string(Frequences(1,10000)));
scf(1);// Ouverture d'une deuxième fenêtre.
clf(1);
plot([0,10000],[p,p],'r');
plot(Frequences,'b+');// + bleu sur les points de coordonnées
// (i, fréquence après le ième tirage). Nous obtenons un nuage de points
// au lieu d'une ligne polygonale. La figure 3 représente la fin du nuage
// après 4 grossissements.
```

```

frequences=Frequences(1,400:400:10000);// Extraction des fréquences
// de 400 en 400 tirages.
scf(2);// Ouverture d'une fenêtre graphique.
clf(2);
a=gca();a.data_bounds=[-0.1,-0.1;25.2,1.1];// à ignorer : réglage
// du graphe.
plot([0,25],[p,p],'r');
plot(frequences,'b*');

```

Commentaires

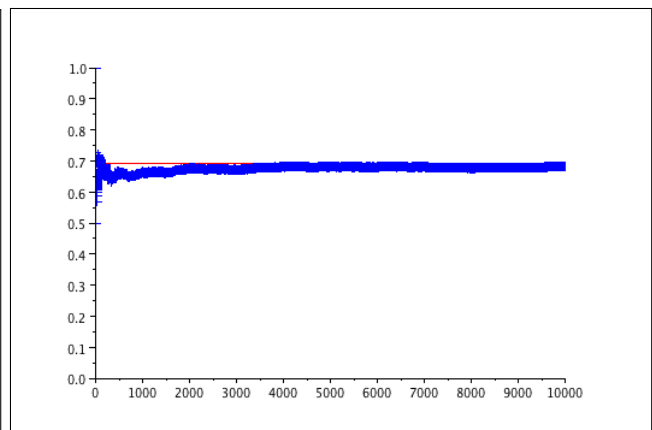
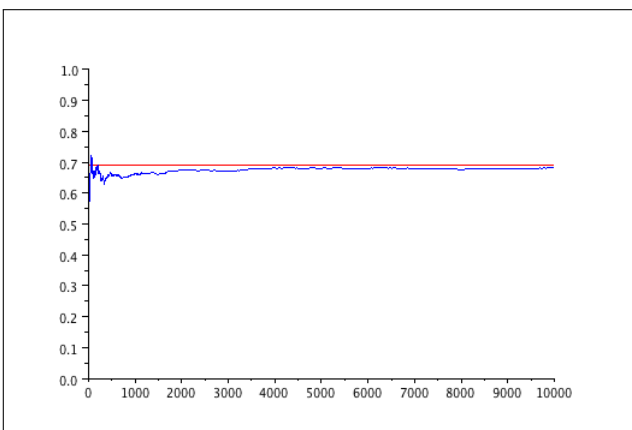
- ✓ Le calcul de p relève du Calcul des probabilités.
- ✓ Il y a une instruction conditionnelle à l'intérieur d'une boucle pour.
- ✓ La condition dans cette instruction conditionnelle (où « Tirages(1,i) » désigne le $i^{\text{ème}}$ nombre au hasard tiré) :

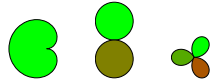
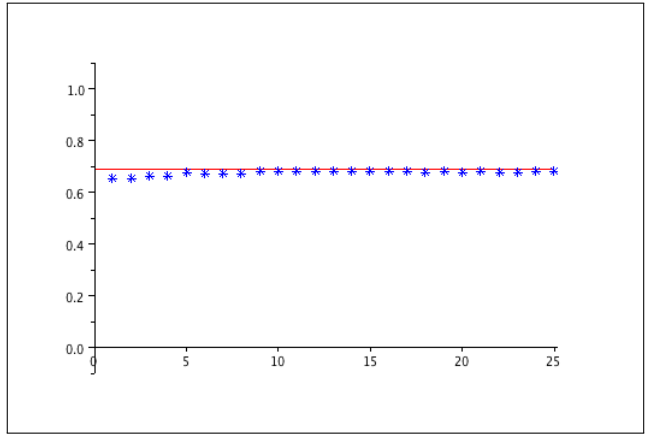
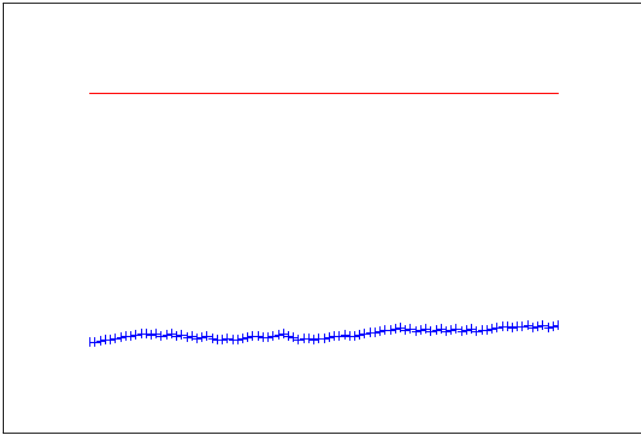
Listing 8 – Instruction conditionnelle

```
(1/7<=Tirages(1,i))&(Tirages(1,i)<=5/6)
```

est la conjonction (symbole « & ») de deux conditions : $\frac{1}{7} \leq \text{Tirages}(1,i)$ et $\text{Tirages}(1,i) \leq \frac{5}{6}$.

- ✓ Dans le script, on ouvre 4 fenêtres graphiques (commandes « scf(0) », « scf(1) », « scf(2) », « scf(3) »). La première fenêtre comprend deux lignes polygonales, dont un segment, obtenues à l'aide de « plot ».
- ✓ Mais ce que l'on veut est l'ensemble des points d'abscisse i dont l'ordonnée est la fréquence après le $i^{\text{ème}}$ tirage, ce que l'on appelle un *nuage de points*.
- ✓ Ceci fait l'objet de la deuxième fenêtre graphique. Chaque point a été représenté par une croix bleue. Le résultat n'est pas satisfaisant. Il y a trop de points (10000). Cela fait plutôt une tache.
- ✓ En faisant plusieurs grossissements successifs de la fin de ce nuage de points, on obtient le graphe de la troisième fenêtre. On y voit bien les points.
- ✓ Une solution acceptable est de tracer moins de points. Dans la quatrième fenêtre figurent seulement les 25 points d'abscisse i et d'ordonnée la fréquence correspondante (sous forme d'astérisques) pour i variant de 400 à 10000 avec le pas 400. C'est suffisant.
- ✓ En résumé, le premier tracé est le plus satisfaisant, bien qu'il ne réponde pas exactement à la question posée.
- ✓ Ces quatre graphes correspondent au même tirage de 10000 nombres au hasard dans $[0, 1]$.





Énoncé 7 [**] : Le point M est-il à l'intérieur du triangle ?

Étant donné un triangle $A_1A_2A_3$ et un point M (par leurs coordonnées), fabriquer une fonction qui indique si M est à l'intérieur du triangle ou non.

Mots-clefs : fonction-scilab, instruction conditionnelle, « input », « exec ».

Soit A et B deux points distincts (on ne peut avoir en même temps $x_A = x_B$ et $y_A = y_B$). Un point M de coordonnées (x, y) appartient à la droite AB si et seulement si

$$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) = 0$$

Si

$$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) > 0$$

M se trouve dans l'un des demi-plans (ouverts) définis par la droite AB ; si

$$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) < 0$$

il se trouve dans l'autre. Pour que 2 points se trouvent dans le même demi-plan, il faut et il suffit, par conséquent, que les quantités associées à ces 2 points comme ci-dessus soient de même signe. C'est ce qui justifie les fonctions-scilab définies ci-dessous. La première retourne 1 si M et A_3 sont strictement du même côté de la droite A_1A_2 , 0 sinon ; la seconde répond à la question : « M est-il à l'intérieur (ouvert) du triangle $A_1A_2A_3$ ».

Listing 9 – M est-il à l'intérieur du triangle A1A2A3 ?

```
// M et A3 sont-ils du même côté de la droite A1A2 ?
function A=memecote(A1,A2,A3,M)// A1, A2, A3 sont trois points distincts.
    A=0;// codage pour 'M et A3 sont de part et d'autre de A1A2'.
    if (((M(2)-A1(2))*(A2(1)-A1(1))-(A2(2)-A1(2))*(M(1)-A1(1)))
        *((A3(2)-A1(2))*(A2(1)-A1(1))-(A2(2)-A1(2))*(A3(1)-A1(1)))>0) then
    A=1;// codage pour 'M et A3 sont du même côté de A1A2'.
    end;
endfunction
// M est-il à l'intérieur du triangle A1A2A3 ?
function A=inttriangle(A1,A2,A3,M)
    c=memecote(A1,A2,A3,M);
    a=memecote(A2,A3,A1,M);
    b=memecote(A3,A1,A2,M);
    if (a*b*c==1) then
        A='M est à l'intérieur du triangle';
    else
        A='M n'est pas à l'intérieur du triangle';
    end;
endfunction;
```

Le second script permet d'entrer les données, d'exécuter la fonction « inttriangle » et d'afficher la réponse à la question posée.

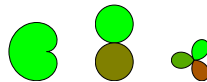
Listing 10 – source scilab

```
// Exécution de 'M est-il à l'intérieur du triangle A1A2A3' ?
// A1, A2 et A3 doivent être 3 points distincts.
A1=input('A1=');// vecteur à des coordonnées de A1.
A2=input('A2=');// de même.
A3=input('A3=');// de même.
M=input('M=');// de même.
exec('/Users/raymondmoche/Magasin_scilab/304/304_inttriangle.sci');
Verdict=inttriangle(A1,A2,A3,M);
afficher(Verdict);
```

Console Scilab

```
-->exec('/Users/raymondmoche/Magasin_scilab/304/304_Interieur-triangle.sce', -1
)
A1=[0,0];
A2=[2,0];
A3=[1,2];
M=[0.5,2.5];

M n'est pas à l'intérieur du triangle
-->
```



Énoncé 8 **[**]** : Ce triangle est-il isocèle ?

Un triangle ABC est donné par les coordonnées de ses sommets dans un repère orthonormé. Écrire un algorithme qui permette d'en déduire qu'il est isocèle (sans être équilatéral), équilatéral ou quelconque (c'est à dire, ici, non isocèle).

Mots-clefs : instruction conditionnelle « if then elseif then end », test d'égalité numérique, « trier », opérateurs logiques « & » et « | », afficher une chaîne de caractères.

Le problème est ici de calculer et de comparer les longueurs des 3 côtés d'un triangle. C'est quelque chose que « scilab » ne sait pas vraiment faire. En effet, parce que les calculs « scilab » ne sont pas exacts (diverses approximations peuvent intervenir), deux longueurs en réalité égales peuvent être considérées comme différentes par « scilab », deux longueurs en réalité différentes peuvent être considérées comme égales. Le premier script néglige ce problème :

Listing 11 – ABC est-il isocèle ? (1)

```
// Saisie des données.
xA=input("xA=");
yA=input("yA=");
xB=input("xB=");
yB=input("yB=");
xC=input("xC=");
yC=input("yC=");
// Calcul de la longueur des côtés.
a=sqrt((xB-xC)^2+(yB-yC)^2);
b=sqrt((xA-xC)^2+(yA-yC)^2);
c=sqrt((xB-xA)^2+(yB-yA)^2);
A=trier([a,b,c]);
afficher(A);// Longueurs des 3 côtés dans l'ordre croissant.
beurk="quelconque";// beurk est une chaîne de caractères.
if ((A(1)==A(2)) & (A(2)<A(3))) | ((A(1)<A(2)) & (A(2)==A(3))) then
    beurk="isocèle";
    elseif (A(1)==A(3)) then
        beurk="equilateral";
end;
afficher('Le triangle ABC est '+string(beurk));
```

Ce script qui ne tient pas compte des approximations de calcul et de représentation des nombres de « scilab » conduit ci-dessous à une exécution fausse.

```

-->exec('/Users/raymondmoché/Magasin_scilab/304/304-isocèle1.sce', -1)
xA=0
yA=0
xB=2
yB=0
xC=1
yC=sqrt(3)+%eps

      2.    2.    2.

      Le triangle ABC est equilateral

-->

```

- ✓ Le triangle ABC saisi est seulement isocèle, pas équilatéral, alors qu'il est considéré comme tel par l'algorithme. Évidemment, si on le traçait dans un repère orthonormé, comme il se doit ici, on le verrait équilatéral.
- ✓ `%eps` est très petit : c'est en effet le plus petit nombre > 0 reconnu par « scilab »
- ✓ `%eps=2.220446049D-16`, voir un manuel.

Le script suivant peut être considéré comme raisonnable :

Listing 12 – ABC est-il isocèle ? (2)

```

// Saisie des données.
xA=input("xA=");
yA=input("yA=");
xB=input("xB=");
yB=input("yB=");
xC=input("xC=");
yC=input("yC=");
// Calcul de la longueur des côtés.
a=sqrt((xB-xC)^2+(yB-yC)^2)
b=sqrt((xA-xC)^2+(yA-yC)^2)
c=sqrt((xB-xA)^2+(yB-yA)^2)
A=trier([a,b,c]);
afficher(A); // Longueurs des 3 côtés dans l'ordre croissant.
beurk="quelconque"; // beurk est une chaîne de caractères.
if (((abs(A(1)-A(2))<=%eps) & (A(2)+%eps<=A(3))) |
((A(1)+%eps<=A(2)) & (abs(A(2)-A(3))<=%eps))) then
    beurk="isocèle";
    elseif (abs(A(1)-A(3))<=%eps) then
        beurk="equilateral";
end;
afficher('Le triangle ABC est '+string(beurk));

```

En voici une exécution :

```
-->exec( '/Users/raymondmoche/Magasin_scilab/304/304-isocele2.sce', -1)
xA=0
yA=0
xB=2
yB=0
xC=1
yC=sqrt(3)+%eps

      2.      2.      2.

Le triangle ABC est isocele

-->
```

- ✓ Deux nombres sont considérés comme égaux si la valeur absolue de leur différence est majorée par %esp.
- ✓ On a aussi modifié la signification de « <= ».
- ✓ Dans l'exemple considéré, ça marche.

